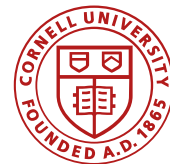


CIFER: Coherent Interconnect and FPGA Enabling Reuse

Integration Meeting July 2019

Princeton University and
Cornell University



**Cornell
University**

CIFER: Coherent Interconnect and FPGA Enabling Reuse



David Wentzlaff

- Professor at Princeton University 
- Was Co-founder & Lead Architect at Tiler
- UIUC Undergrad, MIT SM & PhD
- Research in Manycore, Cloud Systems, Biodegradable Computing
- Runs OpenPiton project

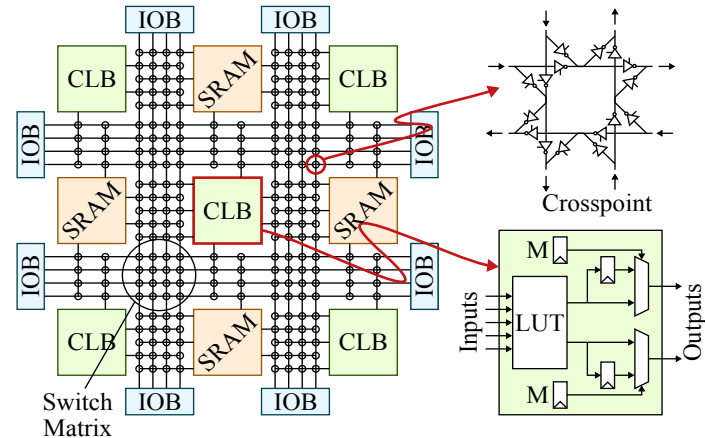
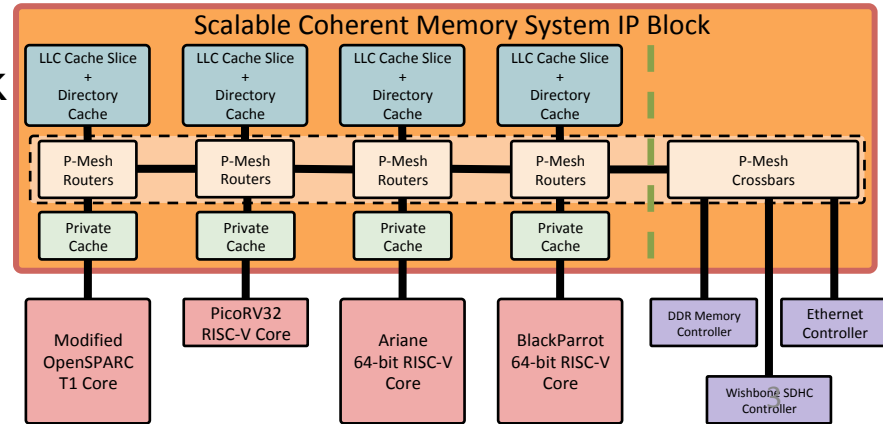


Christopher Batten

- Professor at Cornell University  Cornell University
- UVA Undergrad, Cambridge MPhil, MIT PhD
- Research in Programmable Accelerators, Novel Hardware Design Methodologies

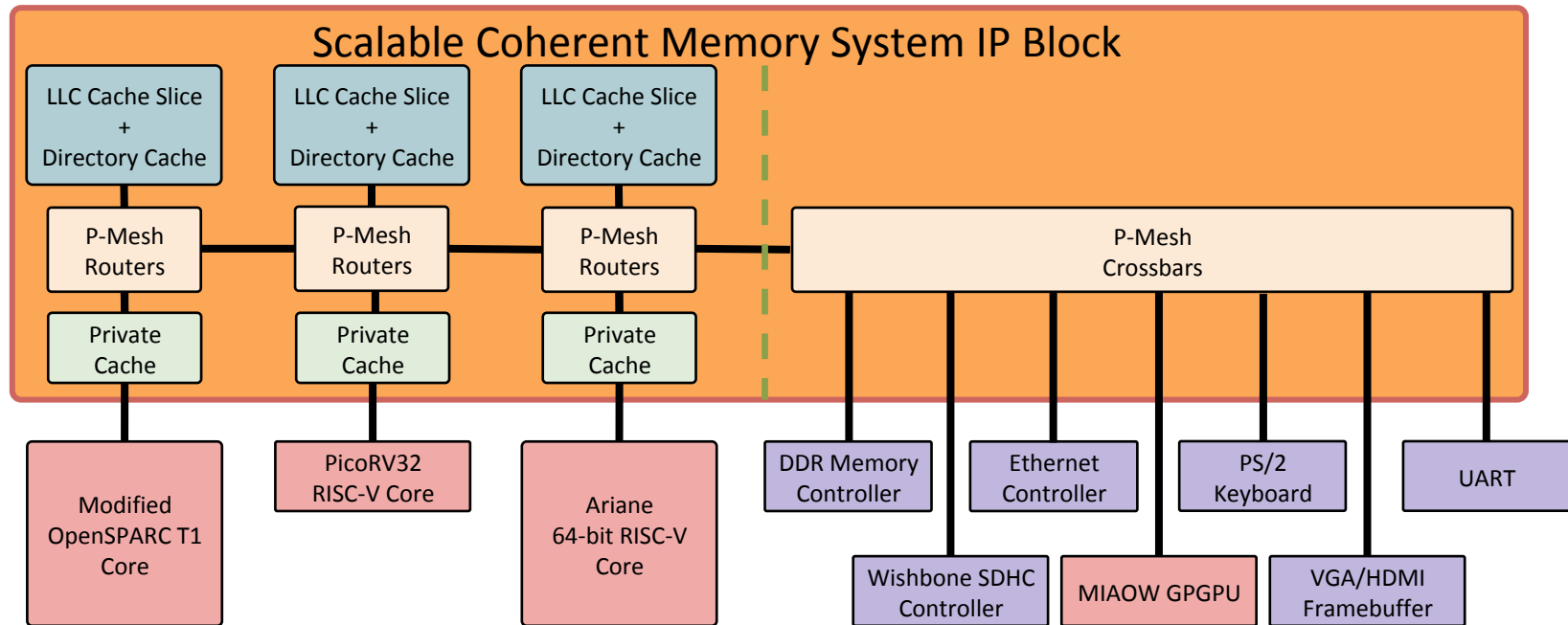
CIFER Project Overview

- Scalable Coherent Memory System IP Block
 - Coherent caches & interconnect
 - Parameterizable on-chip interconnect
 - Supports a large number of cores
 - Leverage extensive work on OpenPiton
 - Interface other POSH cores and IP blocks in a cache coherent ecosystem
- FPGA Fabric IP Block (PRGA)
 - Highly parameterizable design (Python configuration)
 - Generates RTL, configuration logic, Bitstream
 - Fully open source flow
 - Uses, but does not modify, the latest VPR



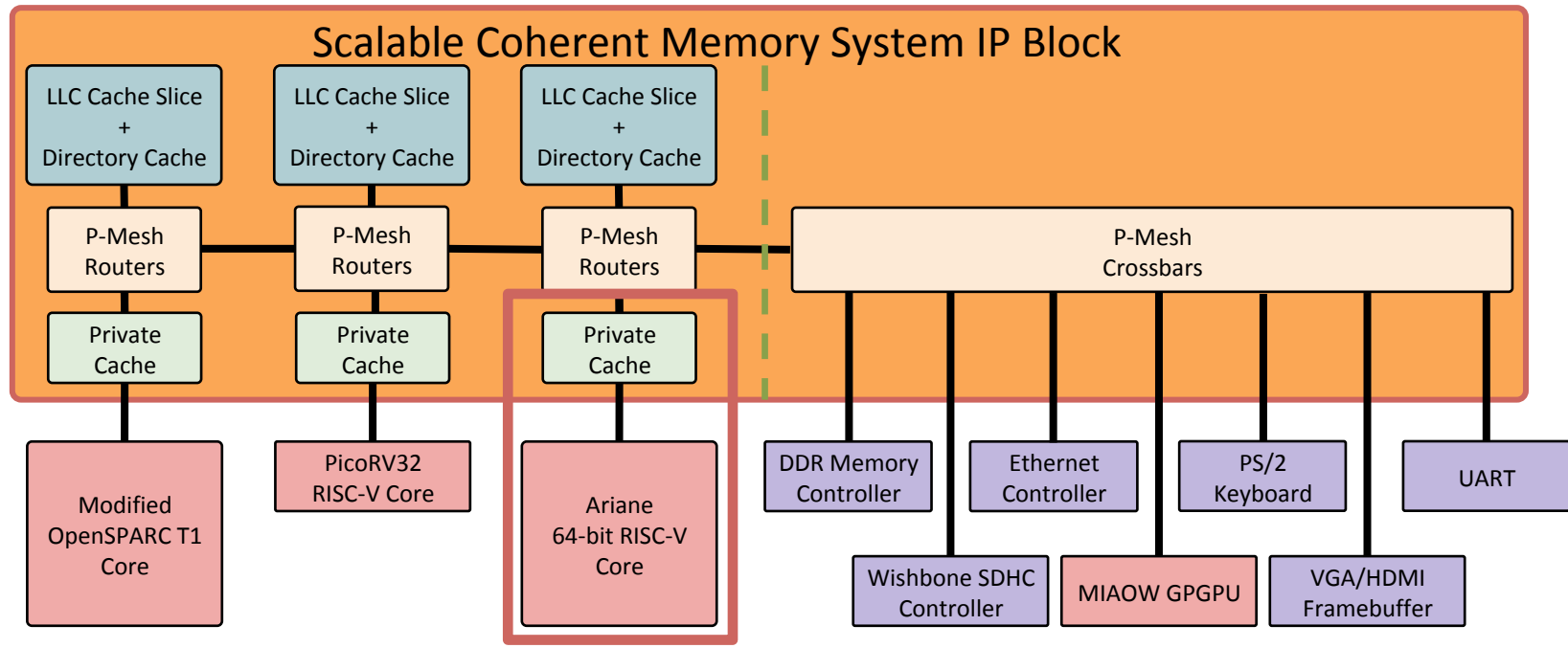
Scalable Coherent Memory System IP Block

- Enable large, open source, heterogeneous SoCs to be created
- Includes coherent caches, directories, and NoC interconnect

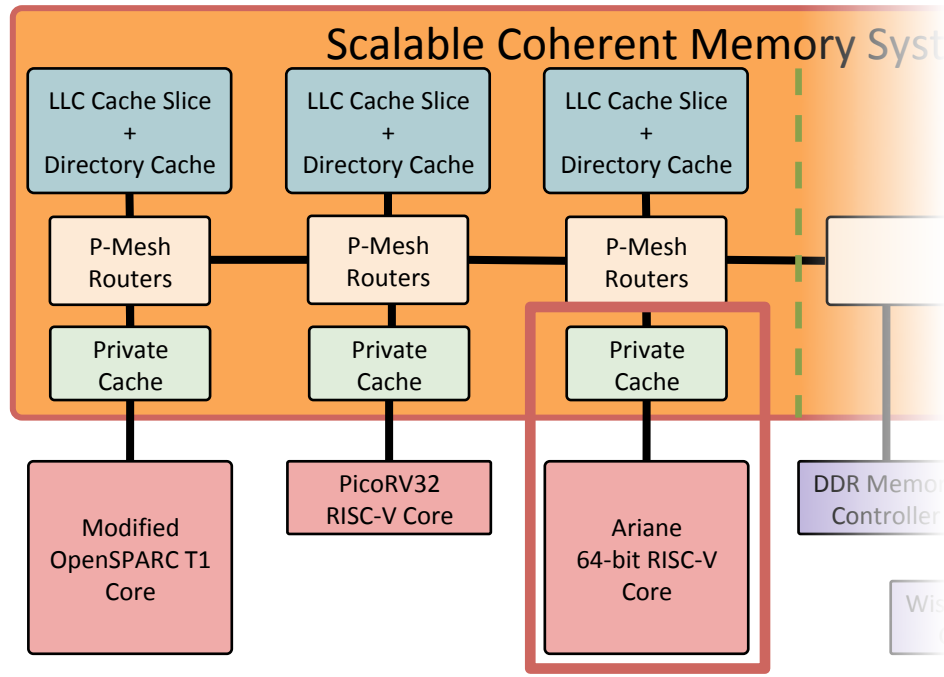


Scalable Coherent Memory System IP Block

- Enable large, open source, heterogeneous SoCs to be created
- Includes coherent caches, directories, and NoC interconnect



Modifying Interface to Support RISC-V Atomic Operations



- Add hardware to support LR/SC in Private Cache
- Add Fetch-and-Op in LLC
- Update interface to support Atomic Operations

Quad-Core Ariane (RISC-V 64-bit) Demo

- Bootloader
- Boot Linux
- Show 4 RISC-V 64-bit cores (`cat /proc/cpuinfo`)
- Demonstrate Tetris
- Demonstrate Vector Addition on {1, 2, 4} cores with speedup

Verifying Scalable Coherent Memory System

- Using memory litmus test cases generated by Jade Aglave's Litmus tool as modified for RISC-V by the CHERI project
 - Running on 4-core RISC-V FPGA version of Scalable Coherent Memory System IP Block
- Provides high confidence torture-test cases for memory consistency correctness
- Threads wait for randomly-chosen time before executing critical code
- Can run test cases on FPGA, identify failing cases and simulate to investigate waveforms
- **Identified bug** in our RISC-V atomics implementation of LR/SC (newly added)

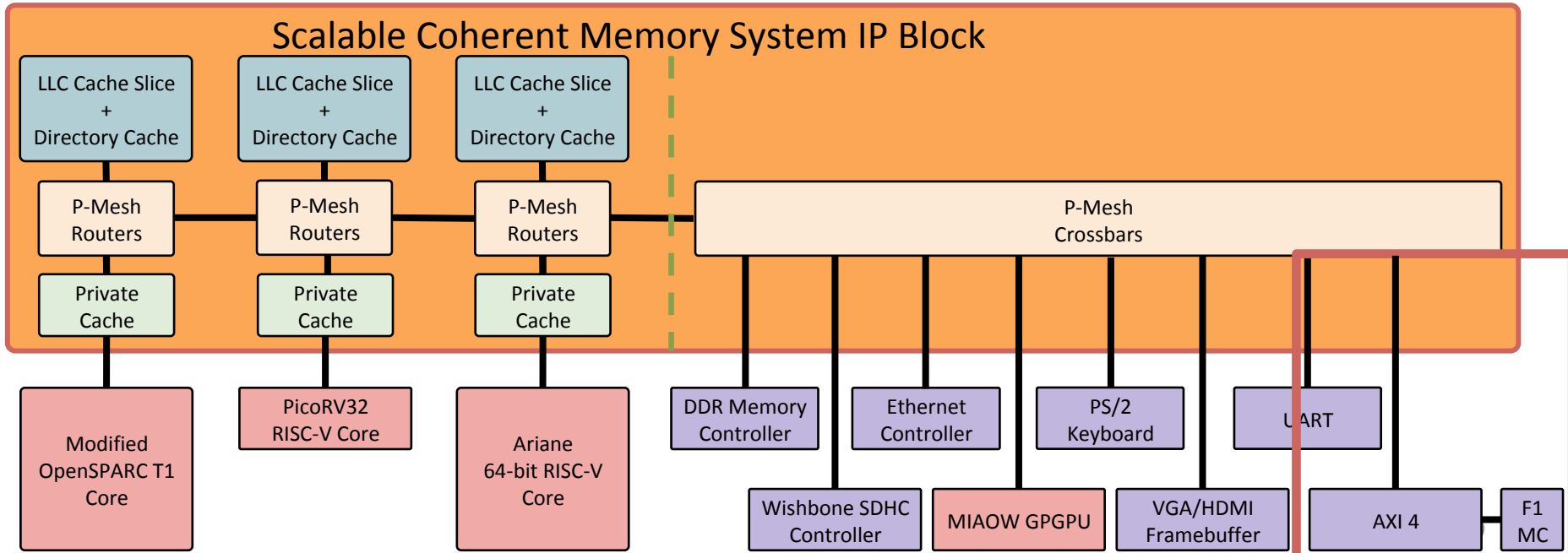
Ex: LR/SC test

```
delay[0] = random();  
delay[1] = random();
```

Hardware Thread 0	Hardware Thread 1
...	...
...	LR.W R1,0(R2)
LR.W R1,0(R2)	ADD R3,R1,1
ADD R3,R1,1	SC.W R4,R3,0(R2)
SC.W R4,R3,0(R2)	...
...	...
...	...
if (0:R4==0 && 1:R4==0 && 0:R1==0 && 1:R1==0) throw; // Should not happen	

P-Mesh to AXI 4 Converter

- Enables connecting more off-the shelf IP Blocks that support AXI
- Can help enable use of AXI memory controllers provided by **Amazon F1 instances**



F1 Demo

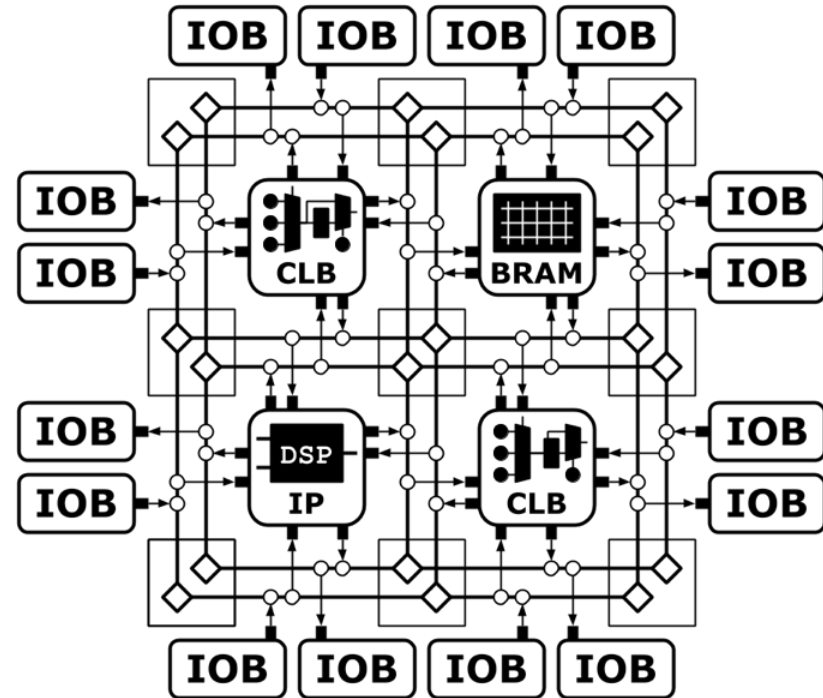
- AWS F1 FPGA configuration load tool
- Copy OS Image into memory
- Start Virtual Serial Connection to FPGA
- Set FPGA Virtual DIP Switches to kick off boot
- Linux Boot

Princeton Reconfigurable Gate Array (PRGA)

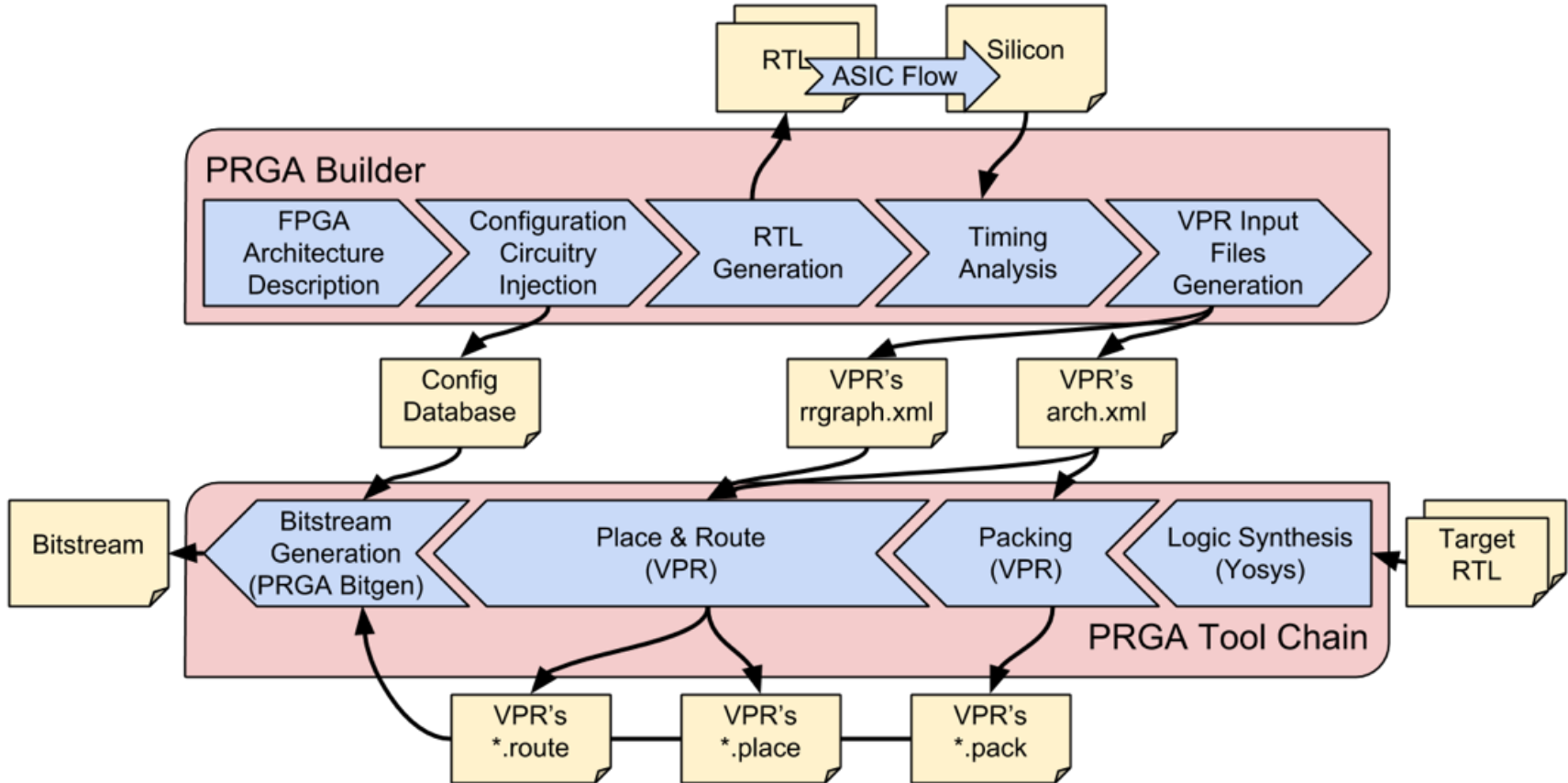
- Open Source FPGA generator
- Highly customizable
- Highly scalable
 - Capable of generating commercial-class FPGAs
- Highly extensible
 - Modularized workflow: replaceable and/or add steps
 - Supports different types of configuration circuitry
- Uses open-source CAD tools (VPR, Yosys), but does not modify other tools
 - Always ready to use latest updates and new features

<https://github.com/PrincetonUniversity/prga>

<https://prga.readthedocs.io>



PRGA Workflow



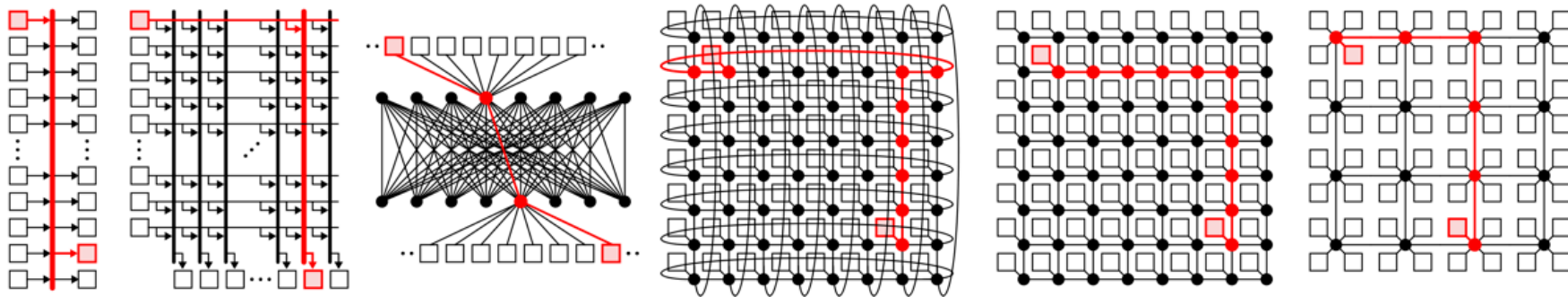
PRGA Updates

- Tested Scalability to larger designs
 - PicoRV32 (RISC-V) core -- DEMO
- Can add arbitrary hierarchy levels
 - Important for physical layout tools
 - PRGA's flexible configuration bitstream order enables independence from VPR tool data structures
- Preliminary automated physical design for PRGA
- **Created novel technique based on automated cycle-free routing to allow FPGA routing to be analyzed by static timing tools**
- Worked with SymbiFlow/Google team to share bitgen code
- Working to integrate PRGA into coherent memory system

PRGA Demo (PicoRV32)

- Build FPGA (Verilog and inputs to VPR)
- Run Behavior Simulation
- Generate synthesis script for Yosys
- Run VPR for Place, Map, and Route
- Generate Bitstream
- Post Implementation Simulation

PyOCN Generator Highlights

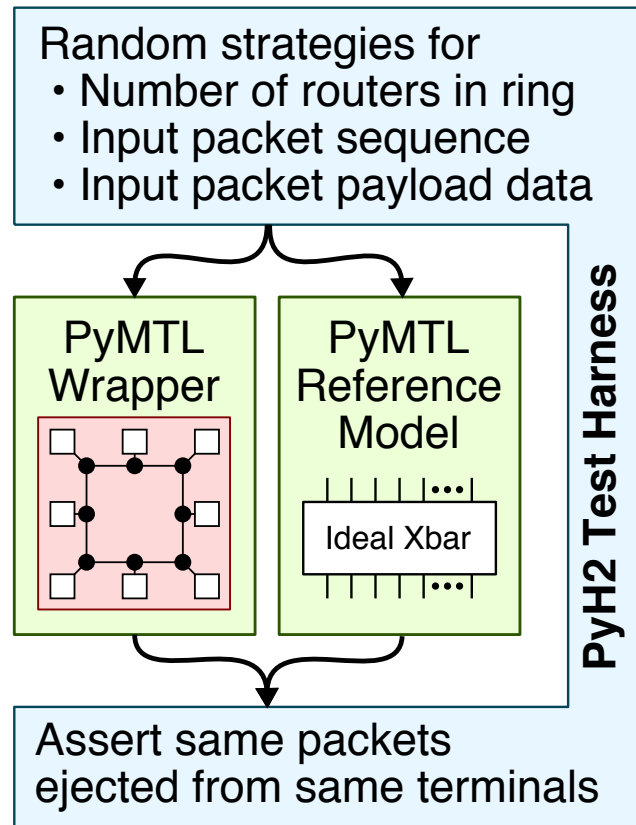
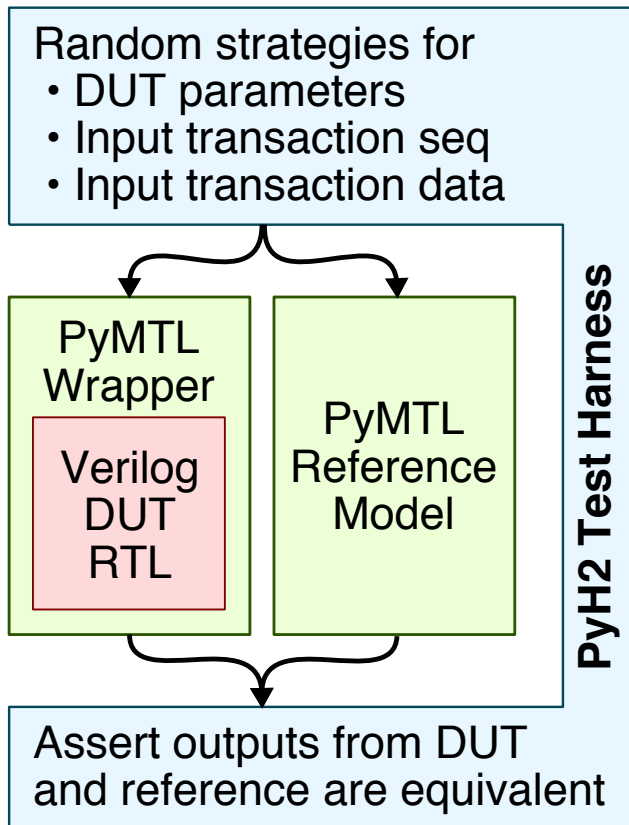


- Migrated generator to PyMTLv3
- New modular and extensible router microarchitecture
- Added support for new topologies and routing algorithms
- Added support for programmable floorplanning and physical design-space exploration of standard-cell-based layouts
- Significant effort on building **PyH2**, a framework for property-based random testing of open-source hardware

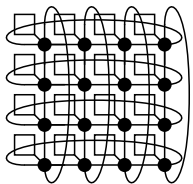
PyH2: Python's Hypothesis for Hardware

Novel Approach:

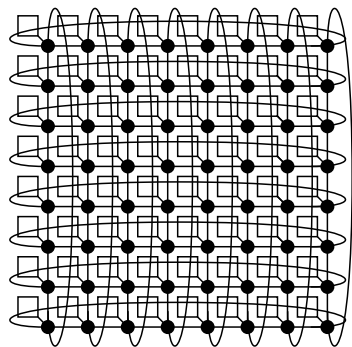
Adapt state-of-the-art open-source software testing methodologies to open-source hardware testing



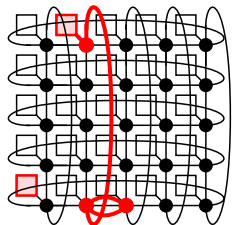
Example of Using PyH2 to Test Torus OCN



- 4x4 torus, minimal routing
- Passes directed tests
- Passes “standard” random tests



- 8x8 torus, minimal routing
- Passes directed tests
- Fails “standard” random test with
 - 100s of cycles
 - 1000s of packets



- PyH2 is used to test torus
- PyH2 spent about ~20 min trying many different network sizes, packets, and payloads

- **PyH2 used auto-shrinking to find a small failing test case which could be debugged in a few minutes**
 - single packet
 - zero payload
 - design to 5x5 Torus

```
always_comb @(*) begin
    if (pkt_dst_x < pos_x) begin
        west_dist = pos_x - pkt_dst_x;
        east_dist = last_col_id
            - pos_x + 'd1 + pkt_dst_x;
    end
    else begin
        west_dist = last_col_id
            + pos_x + 'd1 - pkt_dst_x;
        east_dist = pkt_dst_x - pos_x;
    end
end
```

Demo of Using PyH2 to Test Queue

Verilog Queue Implementation

```

module Queue
#(
    parameter p_data_width  = 32,
    parameter p_nentries = 2,
    parameter c_count_width = $clog2(p_nentries+1)
)(
    ...
    input  logic          enq_en,
    output logic          enq_rdy,
    input  logic [p_data_width -1:0] enq_msg

    input  logic          deq_en,
    output logic          deq_rdy,
    output logic [p_data_width -1:0] deq_msg,
);
...
assign deq_ptr_next
    = ~deq_en          ? deq_ptr :
      deq_ptr == last_idx ? 'd0 : deq_ptr + 'd1;
assign enq_ptr_next
    = ~enq_en          ? enq_ptr :
      enq_ptr == last_idx ? 'd0 : enq_ptr + 'd1;

assign count_next
    = enq_en & ~deq_en ? count + 'd1 :
      deq_en & ~enq_en ? count - 'd1 : count;

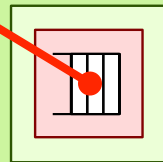
assign enq_rdy = count < max_size;
assign deq_rdy = count > 'd0;
assign deq_msg = data_reg[deq_ptr];

endmodule
    
```

Random strategies for

- Number of entries in queue
- Sequence of enq/deq
- Data enqueued

PyMTL
Method
Adapters



PyMTL
Reference
Model

Python
deque

Assert same data
dequeued from both models

PyMTL Reference Model

```

from collections import deque
from pyrtl import *

class QueueFL( Component ):
    def construct( s, num_entries=2 ):
        s.q = deque( maxlen=num_entries )

    @non_blocking( lambda s: len(s.q)<s.q.maxlen )
    def enq( s, msg ):
        s.q.appendleft( msg )

    @non_blocking( lambda s: len(s.q) > 0 )
    def deq( s ):
        return s.q.pop()
    
```

Using Method-Wrapped Queue

```

def test_auto_tick():
    dut = RTL2CLWrapper(
        QueueVRTL( Bits16, num_entries=2 )
    )
    dut.elaborate()
    dut.apply( ImportPass() )
    dut.apply( AutoTickSimPass() )
    dut.lock_in_simulation()
    dut.sim_reset()

    dut.enq( b16(0xdead) )
    dut.enq( b16(0xbeef) )
    assert dut.deq() == 0xdead
    assert dut.deq() == 0xbeef
    
```

Goals for Integration Exercise


- Integrate Scalable Coherent Memory System IP Block with other POSH IP (processor core, accelerator, or I/O)
- Utilize POSH designs as input designs to PRGA
- Apply PyH2 (Python Hypothesis for Hardware) to other teams' designs identify bugs

Team



 <https://github.com/PrincetonUniversity/openpiton>

 <https://github.com/PrincetonUniversity/prga>

 <https://github.com/cornell-brg/posh-ocn>

The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.